Bluetooth Low Energy (BLE) Identifier Reference

Overview of BLE device identification

At reelyActive (https://www.reelyactive.com), we often get asked questions like *"how much can one know about a person's smartphone from its Bluetooth packets?"* or *"can I recognise subsequent visits of the same device?"* This is intended to be a just-technical-enough reference for anyone to not only find answers to such questions but, more importantly, to understand why.

The following is a primer on radio-frequency identification using BLE. Specifics on the identifiers can be found below.

.

Mandatory identifier

What has 48-bits and comes in three flavours?

The **advertiser address** is the one and only mandatory identifier in a BLE advertising packet. It has the form `12:34:56:78:9a:bc`.

BLE includes a feature which allows this identifier to be either ***public*** (IEEE-assigned) or ***random*** (manufacturer-assigned). Moreover, when random, the device manufacturer is free to change the identifier dynamically, should they so choose. The result is the following three flavours:

- `public static`

    - manufacturer selects unique identifier from a block of IEEE-assigned addresses
    - by definition, this identifier must remain static
- `random static`
    - manufacturer assigns unique identifier at its discretion
    - manufacturer elects for this identifier to remain static
- `random dynamic`
    - manufacturer assigns unique identifier at its discretion
    - manufacturer elects to change this identifier periodically

There *IS* a mandatory flag to indicate whether the address is public or random. However, there *IS NOT* a flag to indicate whether a random address is static or dynamic. In other words:

> the only way to know if a device's *random* advertiser address will periodically change is through observation

COMMON QUESTIONS ABOUT THE 48-BIT ADVERTISER ADDRESS

Question	**`public static`**	**`random static`**	**`random dynamic`**
Can I implicitly recognise the product?	No*	No	No
Can I explicitly associate with a person/product?	Yes	Yes	No
Can I recognise the device again if it leaves and comes back?	Yes	Yes	No
Is the identifier guaranteed to be unique?	Yes	No	No

**it is nonetheless possible to look up the chip manufacturer from this registry (https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries).*

Optional identifiers

But wait, there may be more!

Beyond the mandatory identifier, the standard BLE advertising packet supports up to 31 bytes more payload which may contain additional identifiers.

16-bit UUID

A **16-bit UUID** represents a defined service. It has the form `0x1234`.

The Bluetooth SIG has reserved a block of 512 such UUIDs for member companies, listed here (https://www.bluetooth.com/specifications/assigned-numbers/16-bit-uuids-for-members). It is therefore possible to look up the member company from the UUID. reelyActive maintains this lookup feature in Sniffypedia (https://sniffypedia.org). Notable examples include:

Eddystone

Google's **Eddystone** service uses the UUID **`0xfeaa`**. An open specification, Eddystone includes several flavours that are free for use. These include:

- Eddystone-UID (https://github.com/google/eddystone/tree/master/eddystone-uid#eddystone-uid) which encodes a 16-bit Beacon ID consisting of a 10-bit namespace and 6-bit instance
- Eddystone-EID (https://github.com/google/eddystone/tree/master/eddystone-eid#eddystone-eid) which encodes an encrypted ephemeral 8-bit identifier that can only be resolved by the remote service with which it was registered
- Eddystone-URL (https://github.com/google/eddystone/tree/master/eddystone-url#eddystone-url) which encodes a compressed URL (limited in length by the BLE payload capacity)

Tile

Tile devices advertise the UUID **`0xfeed`**.

Contact Tracing

Devices implementing the Contact Tracing service, a collaboration between Apple and Google in response to the COVID-19 pandemic (specification (https://www.apple.com/covid19/contacttracing/)), advertise the UUID **`0xfd6f`**.

128-bit UUID

A **128-bit UUID** represents a service and/or acts as a device identifier. It has the form `12345678-1234-1234-1234-123456789abc`. Unlike the 16-bit UUID and the 16-bit company identifier used in Manufacturer Specific Data, the Bluetooth SIG does not assign 128-bit UUIDs, leaving vendors and programmers free to choose. Notable examples include:

Fitbit

Fitbit devices advertise the UUID **`adabXXXX-6e7d-4601-bda2-bffaa68956ba`** where the XXXX represents a specific product type such as the Charge HR.

Manufacturer Specific Data

Manufacturer Specific Data is accompanied by a **16-bit company identifier**. It has the form `0x1234`.

The Bluetooth SIG assigns the company identifiers which are listed here (https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers). It is therefore possible to look up the member company from the company identifier. reelyActive maintains this lookup feature in Sniffypedia (https://sniffypedia.org). Notable examples include:

iBeacon

Apple is assigned the company code **`0x004c`** and their devices make extensive use of Manufacturer Specific Data for a variety of functions. While Apple does not openly document most of these functions, the iBeacon specification (https://developer.apple.com/ibeacon/) is a notable exception.

An iBeacon encodes a 128-bit UUID, a 16-bit Major and a 16-bit Minor identifier. Typically, the Major represents a superclass (ex: a physical venue) and the Minor represents a subclass (ex: points of interest within the venue), while the UUID is unique to a beacon vendor or operator, for example:

COMMON IBEACON UUIDS AND THEIR ASSOCIATED VENDORS

iBeacon UUID	Vendor
`f7826da6-4fa2-4e98-8024-bc5b71e0893e`	Kontakt.io
`2f234454-cf6d-4a0f-adf2-f4911ba9ffa6`	Radius Networks
`b9407f30-f5f8-466e-aff9-25556b57fe6d`	Estimote

The above highlights that although all iBeacons must, by definition, transmit Apple's company code (`0x004c`), they are not necessarily Apple devices! In general:

> it is wise to *not* assume that a device advertising a specific company code is necessarily a product of that company.

Local Name

A Local Name is an **ASCII string**. The Local Name is often human-readable (ex: "Charge HR") as the formats described above afford more efficient encoding of machine-readable identifiers. The length of this name is limited by the capacity of the BLE payload.

Additional resources and tools

Best practices for BLE identifiers (diy/best-practices-ble-identifiers/)

Assignment of Bluetooth Low Energy identifiers for interoperability and interpretability

The InteroperaBLE Identifier (interoperable-identifier/)

An open specification by reelyActive to maximise the interoperability of radio-identifiers across protocols, platforms and operating systems.

reelyActive Developers (/)

Browse all developer documentation and tutorials.

For an exhaustive machine-readable list of common BLE UUIDs see Sniffypedia's index.js (https://github.com/reelyactive/sniffypedia/blob/master/index.js)

What's next?

Explore Sniffypedia, check out advlib, or return to the diyActive home page.

Sniffypedia (https://sniffypedia.org)

advlib (https://github.com/reelyactive/advlib#advlib)

Return to diyActive (/)